Programación Python para Machine Learning

Lección 5: Técnicas y métricas de evaluación de modelos de Machine Learning.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 5: Técnicas y métricas de evaluación de modelos de Machine Learning.

1. Introducción

Un aspecto de bastante importancia cuando se quiere poner en producción un proyecto de Machine Learning es el de cuantificar el rendimiento predictivo de los modelos. Es decir, saber cuánto de bueno o malo es un modelo. En una primera aproximación, se podría pensar que la manera ideal de evaluar el rendimiento de un algoritmo sería contar con las predicciones de los datos futuros conociendo su clase. Sin embargo, como parece evidente, ante la imposibilidad de poder contar con esos datos en el momento de la generación del modelo, se hace necesario utilizar técnicas estadísticas que permitan hacer estimaciones precisas sobre el rendimiento de algoritmo.

Objetivos

- Conocer el concepto y la importancia del proceso de evaluación de modelos de Machine Learning.
- Entender los principios del particionado de datos en entrenamiento/test en la validación de modelos.
- Detallar las características de las distintas estrategias de validación cruzada.
- Definir y saber aplicar las diferentes métricas de rendimiento para los problemas de clasificación y regresión

2. EVALUACIÓN DE MODELOS

Todo neófito en el área de Ciencias de los datos tiende a considerar la posibilidad de entrenar un modelo de Machine Learning utilizando el total de los datos que tiene a su disposición, para, posteriormente, utilizar las predicciones del modelo sobre esos mismos datos para valorar su rendimiento. Sin embargo, sería un ejercicio que mucho tendría que ver con "hacerse trampas en el solitario".

En una supuesta situación donde un algoritmo almacene cada observación con la que se entrena, si se evaluara su rendimiento con las mismas instancias, entonces se podría valorar que tendría un rendimiento perfecto. Ahora bien, las predicciones que este modelo realizase sobre nuevos datos estarían muy lejos de ser buenas.

La máxima es clara: la evaluación de los modelos de Machine Learning se debe realizar sobre datos que no hayan sido utilizados para el entrenamiento del algoritmo ni para configurar ningún aspecto de él. Al mismo tiempo, al estar basada en técnicas estadísticas, la evaluación es una estimación del rendimiento, por lo que nunca podrá ser vista como una garantía.

Por último, como consejo práctico, convendría contemplar que, una vez que se haya estimado convenientemente el rendimiento de un modelo, se puede volver a entrenar empleando todo el conjunto de datos, dejándolo listo para su explotación.

A continuación, vamos a ver cuatro técnicas diferentes que podemos utilizar para dividir nuestro conjunto de datos de entrenamiento y crear estimaciones útiles de rendimiento para nuestros algoritmos de aprendizaje automático:

3. CONJUNTOS DE ENTRENAMIENTO Y DE VALIDACIÓN

El método elemental que se utiliza para evaluar el rendimiento de un método de Machine Learning se basa en dividir los datos en diferentes conjuntos: uno para el proceso de entrenamiento y otro destinado a la validación de modelo.

La división se realiza por instancias, nunca por características. Es decir, del conjunto de datos original se selecciona un subconjunto de observaciones con las que nutrirá al algoritmo de entrenamiento. Con las instancias restantes, y sin aportar la variable a predecir, se realizan predicciones. Finalmente, se comparan las predicciones realizadas frente a las reales o esperadas, pudiendo cuantificar el resultado.

El tamaño de la división puede depender del tamaño y las características específicas de su conjunto de datos, aunque es habitual una segmentación 70–30%, 62.5–37.5% o 86.3–13.7%.

Esta técnica de evaluación como tal no se suele utilizar en la práctica para valorar el rendimiento de un método. No obstante, es la base sobre la que se fundamentan todas las técnicas de evaluación existentes. El principal inconveniente de este método es que puede tener una alta dispersión en la precisión.

Diferentes segmentaciones del conjunto de datos para entrenamiento y validación pueden llegar a obtener diferencias significativas en la respuesta del algoritmo.

El módulo de scitkit-learn `model_selection` cuenta con la función `train_test_split` que parte los conjuntos de datos en subconjuntos de entrenamiento y validación de modo aleatorio.

```
from pandas import read_csv
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
import random
import pandas

filename = '../datasets/Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

#missing values
data.fillna(0, inplace=True)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

test_portion = 0.3
seed = random.randint(0,10)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X,
Y, test_size=test_portion, random_state=seed)
model = KNeighborsClassifier(5)
model.fit(X_train, Y_train)
acc = model.score(X_test, Y_test)
print("Acc: ", round(acc*100.0, 4))
```

Ante todo proceso de naturaleza estocástica, hay que tener en cuenta que los resultados pueden variar. Una práctica aconsejable sería considerar la posibilidad de ejecutar el ejemplo varias veces, y comprar los resultados medios y la variabilidad de los mismos a través de la desviación típica.

Esta cuestión se vuelve especialmente importante si se desea realizar la comparación de resultados con otro algoritmo o con el mismo algoritmo configurado de manera diferente. Para asegurarnos de que la comparación es de igual a igual, se debe garantizar que el entrenamiento y la validación se realiza siempre con las mismas instancias de datos.

4. VALIDACIÓN CRUZADA

La validación cruzada es el enfoque por excelencia que se utiliza para estimar el rendimiento de un modelo de Machine Learning. Aporta la ventaja de que es una estrategia con menos varianza que una única división del conjunto en training/test.

A partir de este principio, surgen diversos métodos que pueden ser aplicados.

4.1. Validación cruzada utilizando K-fold

Este método funciona dividiendo el conjunto de datos en *k* partes.

Cada división de los datos se denomina *fold*. El modelo es entrenado con *k*-1 *folds* y se deja el restante reservado, el cual se utiliza para el proceso de validación del modelo.

Esto se repite para que cada *fold* del conjunto, haciendo que todos tengan la oportunidad de ser el conjunto de prueba reservado. Al terminar el proceso iterativo, se obtienen k medidas de rendimiento diferentes, las cuales se concretan en una media y una desviación típica.

El resultado final es una estimación más fiable del rendimiento del algoritmo ante la aportación de nuevas posibles instancias a predecir. La clave está en que el modelo se ha entrenado y evaluado en sucesivas ocasiones con datos diferentes.

La selección del valor de *k* debe adoptar una situación de compromiso donde se permita, por un lado, que el tamaño de cada *fold* de prueba sea lo suficientemente grande como para representar una muestra razonable del problema, pero que al tiempo resulten las suficientes repeticiones de la evaluación para proporcionar una estimación estable del rendimiento.

La comunidad acepta como valor idóneo una *k*=10, aunque en ocasiones también admite que se consideren *k*=3 o *k*=5.

La librería scitk-learn incluye en su módulo `model_selection` la función `KFold` que implementa esta metodología.

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import random
import pandas

filename = '../datasets/Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

#missing values
data.fillna(0, inplace=True)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

k=10
seed = random.randint(0,10)
kfold = KFold(n_splits=k, random_state=seed, shuffle=True)
model = KNeighborsClassifier()
res = cross_val_score(model, X, Y, cv=kfold)
print("Acc: %.3f+/-%.3f" % (res.mean(), res.std()))
```

4.2. Validación cruzada Leave-one-out

La validación cruzada se puede llevar a cabo de forma solo se reserve para validar una única instancia.

Esta variante de la validación cruzada se denomina validación cruzada Leave-one-out (LOOCV). La principal ventaja de esta estrategia es que el resultado final es fruto de un gran número de medidas de rendimiento, obteniendo una estimación más cercana a la precisión del modelo ante los datos futuros.

El inconveniente fundamental viene de que se trata de un procedimiento computacionalmente muy costoso dado que se tienen que entrenar tantos modelos como instancias tenga el conjunto de datos.

La librería scitk-learn incluye en su módulo `model_selection` la función `LeaveOneOut` que implementa esta metodología.

```
from pandas import read_csv
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import pandas

filename = '../datasets/Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

#missing values
data.fillna(0, inplace=True)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

loocv = model_selection.LeaveOneOut()
model = KNeighborsClassifier()
results = model_selection.cross_val_score(model, X, Y, cv=loocv)
print("Acc: %.3f+/-%.3f" % (res.mean(), res.std()))
```

4.3. Validación cruzada de Monte Carlo

En este caso, la validación cruzada se realiza mediante la división de los datos de entrenamiento de forma aleatoria.

En cada iteración, el porcentaje de división entrenamiento-validación es diferente. Se entrena el modelo con el conjunto correspondiente para esa iteración y se calcula el rendimiento del modelo utilizando los datos reservados para el test.

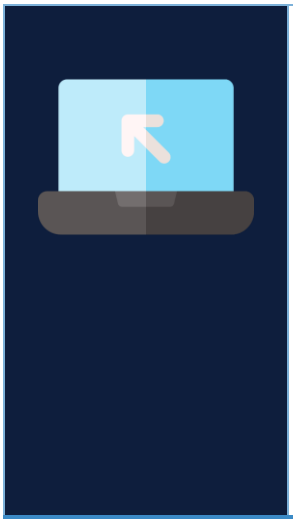
Este proceso se repite un número determinado de iteraciones (100, 500, o incluso 1000 iteraciones). Finalmente, se calcula la media y desviación típica de los valores de rendimiento obtenidos. Hay que considerar que los mismos patrones pueden ser seleccionados más de una vez en algunos casos, o no haber sido seleccionados nunca en otros.

La librería scitk-learn incluye en su módulo `model_selection` la función `ShuffleSplit` que implementa esta metodología.

```
from pandas import read_csv
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
import pandas
import random

filename = '../datasets/Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

#missing values
data.fillna(0, inplace=True)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

k=100
test_portion = 0.3
seed = random.randint(0,10)

montecarlo = model_selection.ShuffleSplit(n_splits=k,
test_size=test_portion, random_state=seed)
model = KNeighborsClassifier()
res = model_selection.cross_val_score(model, X, Y, cv=montecarlo)
print("Acc: %.3f+/-%.3f" % (res.mean(), res.std()))
```

5. MÉTRICAS PARA EVALUACIÓN DE RENDIMIENTO

La elección de métrica para evaluar el rendimiento de los métodos de Machine Learning es determinante, teniendo una influencia directa en la cuantificación y comparación del desempeño. No todas las métricas son válidas en todos los casos. Seleccionar adecuadamente la que corresponde es una tarea que hay que hacer correctamente. La librería scitk-learn incluye en su módulo `metrics` numerosas métricas tanto para problemas de clasificación como para regresión.



Para más información

Se puede encontrar un listado de todas las métricas que implementa scitkit-learn en su módulo metrcis en:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

5.1. Métricas para la evaluación de rendimiento en problemas de clasificación

Existen una multitud de métricas que se pueden utilizar para evaluar el rendimiento de los modelos de Machine Learning en problemas de clasificación.

Probablemente se deba a que son el tipo más común de problema a los que se enfrenta la disciplina. La mayor parte de las métricas están definidas sobre la llamada matriz de confusión.

Los ejemplos en Python que se incluyen para ilustrar el uso de métricas de clasificación han sido realizados sobre el problema binario Indian-Liver-Patient.

Matriz de confusión

La matriz de confusión es la presentación más adecuada pero menos práctica del rendimiento que ofrece un modelo de clasificación para problemas con dos o más clases.

La tabla presenta las predicciones realizadas por el clasificador en el eje x y los valores reales de la clase del patrón en el eje y.

Estructurada como aparece en la Tabla 1, en un problema de clasificación binaria, la predicción puede ser 0 o 1, al igual que las etiquetas de clase reales.

El número de predicciones que son 1 que se corresponden sobre instancias etiquetadas con 1 son los denominados verdaderos positivos (TP). Igual pasa con los 0. Predicción 0 y valor real 0, verdadero negativo (TN).

Las predicciones del modelo que son 1 pero que las instancias son realmente 0, son los falsos positivos (FP); y al revés, predicción 0 y valor real 1, falsos negativos (FN).

		Real	
		1	0
Predicción	1	TP	FP
	0	FN	TN

Tabla 1: Matriz de confusión.

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import pandas, random

filename = '../datasets/Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

test_portion = 0.3
seed = random.randint(0,10)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_portion, random_state=seed)
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
prediccion = model.predict(X_test)
print(confusion_matrix(Y_test, prediccion))
```

Exactitud (Accuracy)

La exactitud (*accuracy*) en clasificación se define como el número de predicciones correctas realizadas en relación con todas las predicciones realizadas.

Es decir:

$$Exactitud = \frac{TP + TN}{nº\ total\ de\ patrones}$$

Esta es la métrica de evaluación más común para los problemas de clasificación, y a su vez, la que peor se utiliza. En realidad, sólo es adecuada utilizarla cuando hay equilibrio en la distribución de los datos de la variable objetivo. Se corre el peligro de caer en la paradoja del falso positivo.

Presta atención

La paradoja del falso positivo
Imagine un problema de clasificación donde el 99% de los patrones pertenecen a la clase A y solo el 1% a la B. Un modelo que siempre predijese clase A para cualquier patrón de entrada tendría una precisión del 99%.

El Área bajo la curva (AUC)

El área bajo la curva (AUC) es una de las métricas más utilizadas para la evaluación en problemas de clasificación binaria. El AUC de un clasificador es igual a la probabilidad de que el clasificador clasifique mejor un ejemplo positivo elegido al azar que un ejemplo negativo elegido al azar. Antes de definir el AUC, entendamos dos términos básicos:

- **Sensibilidad (*Recall*):** Es la tasa de verdaderos positivos (TPR) y se define como TP/(FN+TP). Corresponde a la proporción de puntos de datos positivos que se consideran correctamente como positivos, con respecto a todos los puntos de datos positivos.
- **Especificidad:** Es la tasa de verdaderos negativos y se define como TN/(FP+TN). Corresponde a la proporción de puntos de datos negativos que se consideran correctamente como negativos, con respecto a todos los puntos de datos negativos.
- **Tasa de falsos positivos:** La tasa de falsos positivos (FPR) se define como FP/(FP+TN). Corresponde a la proporción de puntos de datos negativos que se consideran de manera incorrecta como positivos, con respecto a todos los puntos de datos negativos.
- **Precisión:** Se define como el número de verdaderos positivos sobre el número de verdaderos positivos más el número de falsos positivos TP/(TP+FP).

La FPR y la TPR tienen valores en el rango [0, 1]. La FPR y la TPR se calculan con distintos valores de umbral, como (0.0, 0.2, 0.4, ..., 1.00) y se dibuja un gráfico, originando la llamada curva ROC. El AUC es el área bajo la curva de este gráfico correspondiente al valor de FPR frente al valor de TPR en diferentes puntos de [0, 1].

Como es evidente, el AUC tiene un rango de [0, 1], cuanto mayor sea el valor, mejor será el rendimiento de nuestro modelo.

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import random, import pandas

filename = '../datasets/Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

k=10
seed = random.randint(0,10)
kfold = KFold(n_splits=k, random_state=seed, shuffle=True)
model = KNeighborsClassifier()
res = cross_val_score(model, X, Y, cv=kfold, scoring='roc_auc')
print("AUC: %.3f+/-%.3f" % (res.mean(), res.std()))
```

Informe de clasificación

Scikit-learn proporciona un informe muy práctico cuando se trabaja en problemas de clasificación.

Puede ser útil para de un vistazo hacerse una idea rápida del rendimiento de un modelo utilizando una serie de métricas.

En concreto, la función `classification_report()` muestra la precisión, el *recall*, el f1-score (médica harmónica entre la Precisión y el Recall) y el número de ocurrencias reales de cada clase (*support*).

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
import pandas, random

filename = '../datasets/Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
'sgot', 'tp', 'alb', 'ag' ,'class']
data = read_csv(filename, names=col_names)

X = data[data.columns[:-1]]
X = pandas.get_dummies(X, prefix=['gen'])
Y = data['class']

test_portion = 0.3
seed = random.randint(0,10)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_portion, random_state=seed)
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
prediccion = model.predict(X_test)
report = classification_report(Y_test, prediccion)
print(report)
```

5.2. Métricas para evaluación la de rendimiento en problemas de regresión

Existen también un gran número de métricas que se pueden utilizar para evaluar el rendimiento de los modelos de Machine Learning en problemas de regresión. Se puede encontrar un numeroso grupo de estas métricas implementadas en scitkit-learn, específicamente, en su módulo `metrics`.

Los ejemplos en Python que se incluyen para ilustrar el uso de métricas de regresión han sido realizados sobre el problema conocido como *housing*.

Raíz del error cuadrático medio (RMSE)

El error cuadrático medio (MSE) proporciona una idea de la magnitud total del error cometido por el modelo regresor sobre el conjunto de datos.

La raíz del error cuadrático medio (RMSE) convierte las unidades del error en las unidades originales de la variable a predecir.

Se trata de una métrica bastante significativa para la descripción y presentación del error. Se le considera la métrica por excelencia utilizada para regresión.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y - \hat{y})^2}{N}}$$

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import random, numpy as np

filename = '../datasets/housing.csv'

col_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=col_names, sep=" ")

X = data[data.columns[:-1]]
Y = data['MEDV']

test_portion = 0.3
seed = random.randint(0,10)

kfold = KFold(n_splits=10, random_state=seed, shuffle=True)
model = LinearRegression()
res = cross_val_score(model, X, Y, cv=kfold,
scoring="neg_mean_squared_error")
print("RMSE: %.3f+/-%.3f" % (np.sqrt(abs(res)).mean(), res.std()))
```

6. PUNTOS CLAVE

En esta lección hemos aprendido:

- Valorar la verdadera potencialidad de Python y los módulos disponibles para el desarrollo de proyectos de Machine Learning.
- Definir el concepto y la importancia del proceso de evaluación de modelos de Machine Learning.
- Entender los principios del particionado de datos en la validación de modelos y realizar con las herramientas de scikit-learn la división entrenamiento/test .
- Especificar las variantes de las distintas estrategias de validación cruzada.
- Saber aplicar las diferentes métricas de rendimiento para los problemas de clasificación y regresión.